CPE 349: Counting Inversions

Consider the following problem that arises in analyzing rankings to find people with similar tastes. (Once it has been decided two people have similar tastes then their previous choices can be used to make recommendations based on what the other person has liked.) So given a ranking and a database of rankings from other people, how do you find people who have similar rankings?

We will simplify by assuming everyone is ranking the same n things – say books. A natural approach would be to label the books with integers and label them in the order of the first person's ranking. Then reorder these labels by the ordering of another customer's preference. Then count the number of books that are "out of order", in the second person's ordering relative to the first person's ordering(ranking).

Consider the following problem, given n positive integers a_1, …, a_n, we want a measure of how scrambled the numbers are. Think of these numbers as the rankings of the second person. We will do this by counting the number of inversions. **The definition of an inversion is given on page 138 Exercise 11 of Levitin in the problems for Chapter 4.**

Your assignment is to develop and implement an algorithm to count inversions whose worst case computational complexity measured by the number of comparisons is $\Theta(n \log n)$. You may consult the web and other students for ideas. However all code most be your own.

0. Get an idea of how to solve the problem. **MergeSort is a good place to start**. Test it by hand on small problems. Finally write the pseudo code for your algorithm to count inversions.
1. **Implement your algorithm recursively in Java using Divide and Conquer**. The algorithm must be clear and as simple as possible. Its worst case complexity should be in O(n * log n)
2. Determine the recurrence relation that describes the number of comparisons of the entries of the array that contains the permutation as a function of the length of the array.
3. Solve the recurrence relation by back substitution.

Deliverables:

1. Class **Inversions.java** must contain a method that returns a non- negative integer, **public static int invCounter (int [])**.
2. The method should return the number of inversions in the array. The numbers in the array will be a subset of {1, 2, …, 999}.
3. The method invCounter **must** be designed to clearly indicate the divide step, the conquer step, and the combine step. **Your code must be easy to understand and contain good comments**. The conquer step may (but does not have to) be a separate method. If you do use a separate method, make sure your comments make it clear in the method invCounter where the call is, what parameters are passed and what is returned.
4. Submit your Java code for the class **Inversions.java** to PolyLearn.
5. At the same time submit a pdf file, **CountAnalysis.pdf** , that contains
   a. The recurrence relation for the number of comparisons of array entries needed by the algorithm to count the inversions.
   b. Show the derivation (**using back substitution**) of **the closed form** solution for the number of comparisons.

Input to the method:

```
6 4 3 1
```

Returned by the method:

```
6
```

Input to the method:

```
2 3 8 6 1
```

Returned by the method:

```
5
```

As usual you can assume the input is as specified.